Text Compression Evaluation

Ross Arnold

November 1, 1996

Contents

List of Tables

List of Figures

Abstract

Approaches to the evaluation of text compression algorithms have generally been not as convincing as they might be. There are concerns that algorithms might be being fine-tuned for performance on a particular data set. The trend is to evaluate performance on a small number of arbitrarily selected files. We show that this approach can lead to inconsistent results, and suggest alternatives. One alternative method using a set of test files with desirable characteristics is discussed in detail. We show that a relatively small set of files can legitimately be used, as long as some care is taken in their selection.

Keywords

Text Compression, Data Compression, Evaluation, Corpus

1 Introduction

Data Compression is a set of techniques aiming to represent information with less data. It is used in the areas of information storage and transmission. *Text Compression* is a subset of data compression dealing with *lossless* compression. The term lossless indicates that the original data is able to be reconstructed exactly rather than approximately. This report is confined to the evaluation of text compression techniques only.

A large number of compression algorithms have been proposed and implemented. It is possible to evaluate and compare these based on a number of criteria. The principal three of these are the amount of compression, the amount of memory required, and the speed. These factors can be studied analytically or empirically; this report deals exclusively with empirical studies, which can complement and confirm analytical results.

Comparison techniques are likely to be useful to people in a number of different areas. Researchers in compression need to determine whether proposed refinements are of practical use. Developers of new algorithms need to compare performance with that of previously developed algorithms. Users of programs would like to compare them in order to select one most suited to their requirements. Likewise developers of systems incorporating some compression techniques would like to choose that which best suits their system. For example, real-time compression applications are likely to prefer *symmetric* algorithms, which have similar compression and decompression speeds (Bryan, 1995).

In all of the above applications it is important to have reasonable confidence in evaluation figures and performance comparisons. Here *confidence* indicates that published results from one experiment should be useful in some wider context, and should in principle remain valid for a variety of kinds of data that could be encountered.

In practice, it is impossible for a compression algorithm to devise an evaluation method which suits this requirement exactly. The number of possible inputs is infinite. For example, even if all files were restricted to a length of 10 bytes with each byte drawn from the 128-symbol ASCII alphabet, there are 128^{10} or about 10^{21} possible files. In practice there is no such restriction in input size. If all possible files are considered, files can not be decreased in size on average — they will be expanded on average. However, there is a significant semantic restriction — we tend to compress useful data, and useful data tends to have some pattern. This excludes most of the possible files from consideration because they are essentially random and thus unlikely to be used as input for compression algorithms. Nevertheless, the number of possible inputs still remains very large.

The problem is how to produce a result which reflects results on all possible future inputs. One solution would be to take a random sample; the technique of statistical representation of a large population by a relatively small random sample is well understood. However, this encounters obvious practical difficulties. The population is constantly changing as files are created, modified, and deleted, and the population is widely distributed across many different computer systems. Even if these could be overcome, data and information is essentially dynamic — there is no way of knowing whether (for example) a particular file type which may be widely used in the future is represented in the sample.

The solution used here is to select a representative sample from some of the available files. The *Canterbury Corpus* is developed in this way. The popular Calgary corpus, which has been widely used for text compression evaluation, is compared with the *Ghost Lake* corpus. The *Ghost Lake* corpus was also selected arbitrarily; this comparison attempts to determine the possible effects of such an arbitrary selection.

The remainder of this report surveys current comparison techniques, and goes on to discuss various issues related to the evaluation of text compression techniques. These are related to the development of the Canterbury Corpus. The method and results for the development of the Canterbury Corpus are then discussed.

2 Survey of Compression Comparison Techniques

Data compression techniques can evaluated both analytically and empirically. Empirical methods involve implementing the proposed algorithm(s) and applying them to various data. This section examines data that has been used in the past by researchers.

A review of the past three years' papers (1994-1996) presented at the annual IEEE Data Compression Conference gives an indication of the test data currently in use. This data is summarised in Table 1. The table shows that domain specific or one-off data was used in about 31% of the studied publications which quoted experimental results. Note that a number of papers use more than one type of test data. These have been counted twice, so the totals quoted are greater than the actual number of papers surveyed. Domain specific data includes cases where a compression algorithm is being developed for a specific kind of data, for example a representation of DNA chains. Thus the algorithm is only tested on that particular data type. The use of domain specific data is reasonable where a compression technique for that particular domain is being developed. However, using one-off data, where a few files are collected locally, is less justifiable when evaluating general compression algorithms. This points to a need for a widely available corpus to satisfy requirements for particular file types, in particular more common file types, while encouraging repeatability of experiments.

	1994		1995		1996		TOTAL	%
	papers	posters	papers	posters	papers	posters		
No test data used	12	51	10	40	7	20	140	
Data referred to but not identified:								
— lossy	4	6	3	5	1	3	22	11.8
— lossless		2		1	2	5	10	5.4
Data not used by other authors	1	9	11	6	11	4	57	30.6
Lena	5	4	9	5	8	4	35	18.8
Calgary Corpus	3	1	2	2	3	3	14	7.5
USC	2	1	5	4	2	2	16	8.6
Simulated Source	3	1	1	2	4		11	5.9
CCITT	2		2	1	3		8	4.3
Miss America Video		1		1	1		3	1.6
CT medical	1			1	2		4	2.2
Football Video	1				1		2	1.1
JPEG Standard					2		2	1.1
JBIG Standard			1		1		2	1.1
TOTAL	49	76	44	68	48	41	326	100

Table 1: Test data used in Data Compression Conference papers and posters.

For repeatability of experiments, it is important that others are able to obtain the same data. For this reason, data sources should always be published with the results. This is a nontrivial problem, as computer data is essentially transitory, especially when considered over periods of years or decades. Old versions of test data may no longer be available. Phrases such as "The test data is available on request" (Kiselyov & Fisher, 1994) are likely to be of little use in attempts to repeat the research in ten or even two years time. The authors may no longer be contactable or they may have misplaced the data in question. Likewise, citing a particular world wide web address (Slyz & Neuhoff, 1996) as a data source is likely

to be less useful in the long term. Files are often moved or deleted, and domain names can disappear or change.

It can be noted from Table 1 that about 17% of experimental results gave no usable indication of their data (Frey & Hinton, 1996; Inglis & Witten, 1994; Ferens & Kinser, 1995, for example). Others reference literature in which the test data was first used, or directly cite the source of their data.

A more acceptable alternative is the use of widely available published data. Equally important is the accurate citation of the data source. For example, "The audio sample is the entire piece (2 mins, 27 secs) *Every Day I Have the Blues* by B.B. King. Captured from track 1 of the CD *B.B. King Live at the Cook County Jail*, MCA Records" (Shamoon & Heegard, 1994) is preferable to "audio sample from a popular music CD" (Ferens & Kinser, 1995).

Citing data which has already been widely used is useful for comparability of results. An example of this is the image "Lena", used in 19% of cases where experimental data was used, or 22% of cases using lossy data. This image is considered a good test of lossy methods because it provides a variety of shades and textures. However, it is also a good example of the problems which can arise. At least three different names (Lena, Lenna, and WomanHat) have been used for this image. Also, it appears that different versions of the image exist. For example, at least two different scans of the image, at different resolutions and in colour and grey-scale versions, are available. Finally, the image is not public domain, having been first published in a magazine. From Table 1 we see that there may have been a number of copyright violations.

The data shows that the Calgary Corpus, which has been available by ftp for about six years, is the only lossless standard which was used more than once in the papers surveyed. This indicates that this format, consisting of a small number of files, is acceptable to the text compression community. The Canterbury Corpus, presented later in this paper, is of similar format. This suggests that the Canterbury Corpus has a good chance of similar acceptance.

3 Issues in Evaluation of Compression Techniques

There are several interesting and important issues that are worth consideration when evaluating data compression algorithms. Some of these are discussed in the following section.

3.1 Versions of algorithms

In practice, algorithms are evaluated by applying particular programs which implement those algorithms. For example, a variant of the *LZW* method is represented by `compress`. Like any other program, compression programs are often altered over time as new versions are released. These different versions are likely to deliver different compression performance, as a principal reason to change a compression program is to further optimise it for speed or amount of compression. Even a change to the default parameters can have an effect. It is therefore important to consider which version of any program is being evaluated. Merely stating that *compress* was used is insufficient information; *compress revision 4.0, 30 July 1985* is an improvement.

This relates directly to the principle of *repeatability* of experiments. Enough information about programs used should be noted for the experiment to be repeated by others. Ideally longer time frames, measured in decades rather than months, should be considered. This focuses attention on describing both the algorithm and the program, as much as possible, within the evaluation report, rather than using extensive reference to external sources. Any external sources used should be examined with regard to longer time frames, in which references to particular WWW pages (Slyz & Neuhoff, 1996) or particular people (Kiselyov & Fisher, 1994) become less useful.

It may also be noted that the experiment is dependent to some extent on the reliability and documentation of the program author. A not uncommon problem in any software is that of the program's operation differing from its documentation.An implicit assumption is that the program implements the algorithm as claimed. There is potential for this to invalidate results, or at least make them less useful.

These factors affect the fundamental reliability of the experimental evaluation. Although it is the algorithm one generally wishes to evaluate, it is the *implementation* that is actually tested. If the implementation used in the evaluation is not recoverable, the evaluation is not repeatable. If the implementation misrepresents the algorithm, the results are misleading.

3.2 Honesty, fraud, and misunderstanding in compression

In 1992 WEB Technology claimed that by use of their product "virtually any amount of data can be squeezed to under 1024 bytes." This could be done because their algorithm could "compress its own output multiple times". This can easily be shown to be impossible. The number of compressed representations would be fewer than the number of possible inputs, so lossless decompression is not possible and the compressed data can't be restored to its original form. The product was never released (faq, 1996).

This type of problem is scattered through the history of compression. As well as the spurious recursive application argument, there have been claims of the ability to compress any (random) data; attempts to hide the supposedly compressed data; of failure to include all required information (such as dictionaries) in compression figures; patents for impossible methods; and reporting of misleading figures obtained from testing on only a very small number of inputs. Other unusual strategies such as detecting files from the Calgary Corpus (Bell *et al.*, 1990), storing the file in

another computer via the Internet, or searching the local file system for the original file when decompression is requested, can easily be imagined.

Some of these can be quickly discredited. The OWS and WIC programs (faq, 1996) hid the original data in unused disk blocks. The "compressed" file consisted of a reference to the location of the hidden data. While this example of compression fraud was relatively easy to detect, other cases have been more subtle.

This can be the result of intentional misrepresentation, but is at least as likely to come from an imperfect understanding of the issues involved, or failure to perform reasonable tests. One less extreme example is provided by Abrahamson (Abrahamson, 1989). A small number of files were used for evaluation, and compression figures are quoted to four significant figures. Differences of 0.01% are not informative for a single arbitrarily selected file. To present results to this level of precision might lead readers to think there is some significance. This is often done in the evaluation or comparison sections of papers which discuss compression algorithms.

The Abrahamson paper also gives results on artificially generated files which are especially suited to the compression methods discussed. For example, one such file "contains 10000 copies of the string *aaaabaaaac*." Although this can be justified as a demonstration of the algorithm's ideal performance, it is presented in the same way as results on the real files ; the 87.91% saving reported is misleading because the file is highly unusual.

Another example of an unintentionally misleading claim can be found on the *zlib*[1] home page on the world wide web:

> "Unlike the LZW compression method used in Unix compress(1), the compression method currently used in zlib essentially never expands the data. (G.Roelofs, 1996)"

A link promising additional information on this point leads to the explanation that this system avoids excessive expansion by including a flag followed by the original file verbatim if the file would have been expanded. This is fine, but the original statement was certainly open to misinterpretation.

A similar example can be found in Wheeler (1995).

> "The speed is slow but the compression is close to optimum."

Although "optimum" is later defined to mean "the performance of PPM*", this statement is definitely misleading.

A certain amount of skepticism seems to be justified in investigations of the field of compression. Intentional dishonesty is perhaps less likely to be encountered than "honest mistakes". This is due to the ease with which evaluations can be repeated independently, exposing any attempt to publish fraudulent data. However, repeating evaluations is not always straightforward (see section 2). Unintentional deception has appeared; researchers should be wary of this.

More attention given to rigorous evaluation of compression programs would enhance the credibility of publications. In particular it is important to state what is being done — a general evaluation and comparison, or an example of performance on one or two files which perhaps yields better figures than might be expected in general. The relatively small additional effort required to perform a more general evaluation (even if only including single representatives of a dozen different file types) is worthwhile in terms of credibility and interest.

In an attempt to avoid or minimise some of the above problems, we intend where possible to carry out controlled evaluations of programs on the Canterbury Corpus in order to provide an independent evaluation. This is as an alternative to

[1] A variant of *gzip*.

distributing the corpus and collecting results. Of course, this is not guaranteed to eliminate fraud, errors, and misrepresentation. The only way to be really certain of a result is to produce an independent *implementation* of an algorithm, enabling confirmation that nothing untoward is done by the program.

3.3 Measurement

Experiments aiming to evaluate compression algorithms inevitably involve measurement. The units in which results are measured affects their usefulness.

In this section we discuss various ways in which speed and compression have been measured, and suggest preferable units for these quantities.

3.3.1 Compression Measurement

Recently published papers on compression use a variety of units for measuring compression. Units used include the following.

- Bits per character (bpc), for example 2 bpc.
- Bits per pixel, for example 0.25 bits per pixel.
- Bits per symbol, for example 2 bps.
- Multiple of compression achieved by some benchmark compression utility.
- Ratio of uncompressed file size to compressed file size, such as 4:1 or 4.
- Ratio of compressed file size to uncompressed file size, for example 25% or 0.25.
- Percentage reduction or percentage saving, for example 0.75, 75% or simply 75.
- "All results are given in entropy" (Wu, 1996), for example "2".
- Give the size of the file(s) before and after compression, as in (500 bytes, 125 bytes).

The last listed is probably the easiest to understand, but is cumbersome because of the space needed to represent the requisite large numbers. This measure is difficult to use in comparisons where different files are used, and difficult to take in at a glance.

Entropy or bits per character is the least ambiguous because it is less dependent on the raw data. Bit per character/pixel/symbol are good for particular file types, but can't be applied to all files. They have the advantage of remaining valid across different source representations (9-, 8-, or 7-bit text for example).

Ratios can be used for any file. However, there may be some initial confusion as to the direction of the ratio.

In this report, bits per character is used to measure the amount of compression. This measure is unambiguous, relates directly to the way entropy is measured, and gives small numbers which are easy to compare.

3.3.2 Speed Measurement

In order to achieve some degree of independence from the system on which tests are done, it was decided to measure compression relative to the speed of `compress`. Historically, compress has been widely used and should thus be an acceptable basis for comparison. This approach is still not likely to be completely machine independent. Different machine instructions have varying speeds on different architectures, and compression algorithms make use of different sets of machine instructions. However, using the ratio with the speed of `compress` at least offers some hope that experiments may be repeatable on a different system.

The question remains whether to give the "time taken" ratio (seconds) or the "speed" ratio (Kilobytes per second or seconds per kilobyte.) In this report we have used the measure microseconds per kilobyte, because it gives numbers in the order of magnitude of 10^0, and because the numbers increase with faster speeds as would be expected for a measure of speed.

It is considered desirable to measure both compression and decompression speed. Each of these will be more emphasised in particular applications. For example archivers are likely to be used for compression more than decompression, whereas pictures on a CD-ROM are likely to be compressed once but decompressed many times.

3.4 Evaluation Using Files of Known Entropy

One possible method of comparing compression algorithms is to determine how closely they approach the optimum compression. The optimum compression is the *entropy* of the source, and no compression algorithm can do better than the entropy on average (Shannon, 1948).

The best possible compression of files occurring naturally can't be found, although it can be estimated in some cases (Teahan & Cleary, 1996). To overcome this problem we can attempt to generate artificial files whose entropy is known exactly.

Files of known entropy were generated using a finite state probabilistic model. Finite state models were chosen to generate these files because they are easy to specify and are able to implement finite context models as a subset. Markov models have previously been used to generate data for tests (Frey & Hinton, 1996).

One problem encountered in the provision of files with known entropy is that random number generation is required. These are required in order to traverse the finite state model according to the specifications at each state. The most widely used method of simulating random numbers is the *linear congruential pseudo random number generator (LCPRNG)*. Using this method leads to a potential problem in our application area. It is possible in principle for an algorithm to effectively compress the "random" file by representing it simply as the finite state model, the parameters for the pseudo random number generator, and the seed. To do this the algorithm would have to be specialised for compression of files generated in this way. It would need to determine the model by using the model's output, and similarly determine the parameters of the random number generator. This shows that files generated in this way do not really have the intended entropy, because the "random" numbers used have a discernible pattern.

A more likely problem is that the cycle in the PRNG may be short enough to be taken advantage of by a general compression algorithm. LCPRNGs have a known cycle length l, and after generating l numbers, they produce the same l numbers again in the same sequence. If the finite state model returns to its initial state coincidentally with the start of a new cycle, the output will be repeated exactly. Many compression methods are able to exploit this exact repetition.

Notwithstanding the above, for practical purposes it is sufficient to use LCPRNGs. The chance of an algorithm being specifically tailored to compress files generated with this method seems remote — as noted in Section 3.2, we should be able to assume that programs claimed to be usable for general compression are not specialised in this way. The second problem can be avoided by selecting a generator with sufficiently long l, and by ensuring that files generated are less than l bytes long.

Figure 1 is an example of a finite state model. The output will have entropy 0.807 bits per character, for reasonable output length.

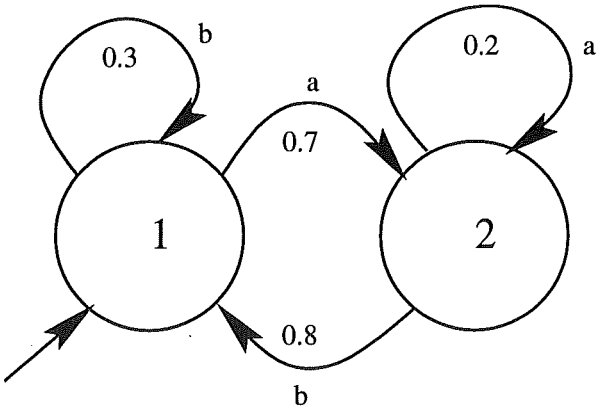

Figure 1: A finite-state model.

```
2
1
1   2   0.7   a
1   1   0.3   b
2   1   0.8   b
2   2   0.2   a
```

Figure 2: Data file used to describe the model in Figure 1

This *reasonable output length* qualifier is necessary because the entropy differs for smaller output sizes. For example if the output length is one byte, the entropy is the same as that of state 1 (the initial state), 0.881 bits. However elementary Markov chain theory indicates that the distribution rapidly approaches steady state. For the example, this is $P(1) = \frac{8}{15}, P(2) = \frac{7}{15}$. Thus the entropy of the output is $-\frac{8}{15}(0.3\log_2 0.3 + 0.7\log_2 0.7) - \frac{7}{15}(0.2\log_2 0.2 + 0.8\log_2 0.8) = 0.807$.

Using this method allows generation of files of any desired entropy. It is easy to specify their characteristics, and also to produce output of any desired length.

A specification for the example finite state model is given in Figure 2. The first number in the file gives the number of states, and the second gives the initial state.

The remaining lines each describe a transition, for example in Figure 2 "**1 2 0.7 a**" indicates a transition from state 1 to state 2 with probability 0.7, the output being the character **a**.

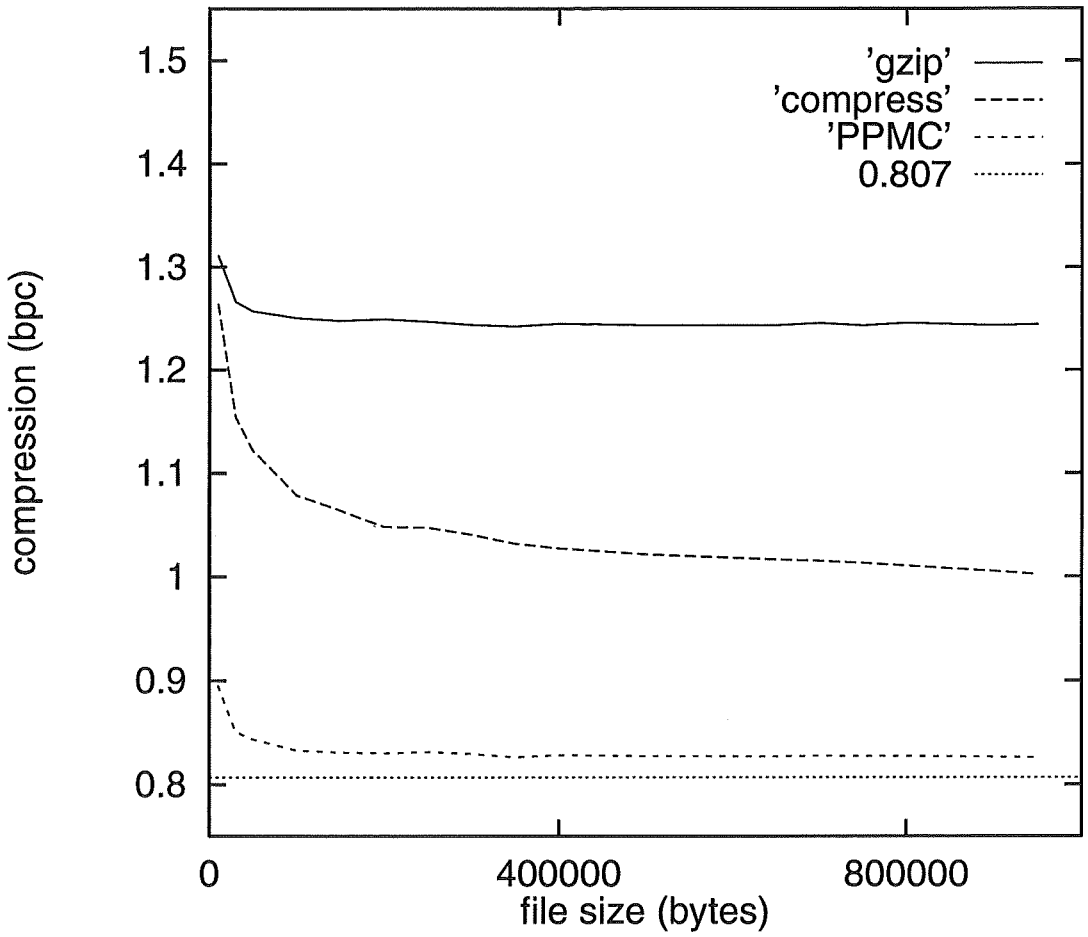

Figure 3: Compression of randomly generated files with a known entropy of 0.807 bits per character.

We implemented a random file generator based on the above scheme. For random number generation, the multiplier $A = 16807$ and the modulus $M = 2147483647$ were used in a standard multiplicative linear generator. These numbers are suggested by Park and Miller (1988) as representing a minimum standard generator, with the advantage of being fast and easy to implement on 32-bit architectures. The cycle length of this generator is M, so it should not be used to generate more than M bytes output. The program reads a description such as the one given in Figure 1 and produces a specified number of output bytes. Several files were generated using this method, which were compressed with a number of programs. Results are shown in Figure 3.

These results give a different ordering to other evaluations of the same algorithms. For example, this method ranks `compress` as better (closer to optimum) than `gzip`. This ordering is almost always reversed for naturally occurring files. This result suggests that this comparison method is not suitable for evaluating general effectiveness, but it adds extra information about the algorithms involved, since

it does not duplicate previous results.

4 Experiments

In this section we discuss empirical generation of a compression corpus. The widely-used Calgary corpus is evaluated, and our goals for development of a new corpus are outlined. The procedure used to generate the new corpus is described.

4.1 Evaluation of the Calgary Corpus

The Calgary corpus is an arbitrary collection of files, described in appendix B of Bell *et al* (1990). These are frequently used as a benchmark for new or improved algorithms. Results are frequently given to two or three decimal places. In some cases, this degree of "accuracy" is required to differentiate between algorithms on the cutting edge of current research, as improvements in compression are becoming smaller.

However, detailed results on the Calgary corpus are not particularly meaningful. If algorithm A averages 3.024 bpc on the Calgary corpus, and algorithm B averages 2.982 bpc, all that can be said is that algorithm A gives better compression on the Calgary corpus. In fact, algorithm A might be optimised for the Calgary file set, but might perform worse in the more general case.

To illustrate this point, we have developed a group of files which parallels the Calgary corpus. Called the *Ghost Lake*[2] corpus, it contains the same number of files of similar type and size. Compressing this corpus, and comparing the results with those using the Calgary Corpus indicates how dependent comparisons are on the particular selection of of files in the Calgary Corpus. This comparison is given in Table 4.1. The same comparative ordering was obtained for these algorithms on these corpuses. However, differences range from 0.07 bpc (compress) to 0.16 bpc (DMC). This result suggests that algorithms which perform similarly on the Calgary Corpus require additional checks to determine which is really "better". We also note that the common practice of giving results on the Calgary Corpus to 2 or 3 decimal places conveys no useful information (apart from performance on the Calgary Corpus).

Algorithm	Compression (bpc)	
	Calgary	Ghost Lake
PPMC	2.28	2.43
bred	2.51	2.65
gzip	2.67	2.77
DMC	2.84	3.00
compress	3.59	3.66
pack	4.98	4.85

Table 2: A comparison of two arbitrarily selected file sets containing files of similar type and size.

4.2 Goals of Corpus Development

It is not possible to generate the ideal test set for compression, because of the large amount of possible unknown data which could be encountered in the future . All we can do is develop a heuristic or empirical approach, and try to make reasonable "guesses" about what should be included.

Only a very small fraction of possible files will ever be used. Even if all files were only eight bytes long, there would be 2^{64} possible files — ten billion for each

[2] A town near Calgary.

person on the planet. The fact that only a few of these are ever used enables us to compress, since compression is a mapping. If we *could* include all possible files, results would not be useful because we would obtain no compression on average. Many of the possible files would be expanded, but a few — hopefully the ones actually used — will be compressed. In some ways, compression can be viewed as gambling on particular files being more likely than others.

In collecting a corpus, we want to select a *representative* sample of these useful files. This would enable us to use the performance of compression algorithms on the corpus to make meaningful general statements about their expected performance, particularly relative to each other

For a number of algorithms, it is desirable to "train" the algorithm on one or more files of similar type. Training on similar data sets reduces the effective entropy of the current data set (Teahan & Cleary, 1996). Therefore we intend to provide groups of similar files with the Canterbury Corpus distribution.

The purpose of these experiments was to

1. Determine appropriate contents for a corpus to be used in evaluation of compression algorithms.

2. Be able to justify the method used to generate such a corpus.

It is desirable for the corpus to be widely used. This facilitates communication within the compression community, because being able to quote results on a known file set for which results are already available for other algorithms makes comparison easy.

In order for the corpus to be widely used, it must be:

- Widely available. The Internet is an ideal medium for this, with ftp and web access to corpus contents important. Possible problems with copyright should be considered.

- Related to the previous point, the contents of the corpus should not be too large. Bandwidth limitations affect potential distribution.

- Widely known. Advertising is important here, as well as informing contacts in the compression community.

- Perceived to be valid and useful. This means that, for example, files of popular and widely-used types should be included. Obscure file types should not be used without good reason. Having the development procedure published is desirable.

- Actually valid and useful. This is where the bulk of the work for this project comes in. It is desirable for the inclusion or exclusion of files to have a solid theoretical and/or experimental foundation. The requirement for generality precludes a comprehensive theoretical foundation. Therefore, we concentrate on making empirical decisions about whether files are to be included.

In order to determine what files should be included, we observe that a principle use for the corpus is to *compare* different compression algorithms. The corpus provides a common reference point. Additionally, accurate *figures* should result from such comparisons. It is desirable that, if we can say that our corpus gives 3.04 bpc for algorithm A, and 2.93 bpc for algorithm B, that this would allow us to make some more general statement. For example, that "B usually gives slightly better compression than A" or "on average, B will compress files of the types given in the corpus slightly better than A".

Given the above requirements for the corpus, it would appear to be useful to establish representative benchmarks for file types, such as a "representative html file" whose results are able to represent all files of that format. It is not possible to find such files, because all files that could possibly be compressed must be known. However, we hope to make a good approximation using a much smaller sample.

First, consider a hypothetical case where there is only one compression algorithm. We could develop empirically a corpus consisting of a single file. This would be done by compressing a large number of files, and thus funding the average compression of this algorithm. We could then simply take the file whose compression ratio is closest to this average, to form our corpus.

Now, consider the case where all possible compression algorithms to be tested are known. With a little more difficulty, we might again obtain a reasonable corpus. Repeating the above procedure for each algorithm, we could try to find a file whose compression is closest to the average in all cases. If we wanted to know the average compression performance of a particular algorithm, we would only have to compress our representative file. This would be just the same as looking up the average compression in a table[3].

Finally, consider the "real" case, where there are a large number of known algorithms and a number of unknown algorithms (yet to be discovered). We could select a representative subset of the known algorithms, and again attempt to find a file whose compression is closest to the average in all cases. Our results suggest that a file set obtained in this way is likely to be applicable to new algorithms, since all the algorithms we use exhibit a similar pattern.

File type	Speed ($\mu s/KB$)	Compression (bpc)
lisp source	1.4	2.66
English Text	2.0	2.48
Binary	4.4	3.59

Table 3: Results of compressing three widely different file sets with the PPMC algorithm. This data shows the speed varying considerably more than the compression.

It seems natural to focus on the amount of compression achieved, rather than on the amount of time taken. The main reason is that speed is less dependent on file type, as shown in Table 4.2. Also, criteria based wholly or in part on the speed of compression methods would be essentially less portable. This is due to the general observation that the amount of compression will remain constant over different platforms, whereas the speed may vary greatly. Attempting to solve this problem by continual reference to a common benchmark (as used elsewhere in this report) may be misleading. The issue of speed measurement has been covered in section 3.3.2.

For a number of algorithms, it is desirable to "train" the algorithm on one or more files of similar type. Training on similar data sets reduces the effective entropy of the current data set (Teahan & Cleary, 1996). Therefore we intend to provide groups of similar files with the Canterbury Corpus distribution.

4.3 A Procedure to develop a corpus

We have observed that there is room for improvement in current evaluation methods. In this section we describe a method to develop a corpus that could be used in more consistent, empirically justifiable evaluations.

In order to achieve the goals detailed in the preceding section, the following procedure was used.

[3] As in the appendix of "Text Compression".

First, a large number of candidate files (about 1000) were identified as being relevant for inclusion in a corpus. These were divided into groups according to their type, such as English text, C source code, UNIX object code, and HTML. Only a small subset of all file types is represented — our aim here was to select a few typical file types rather than to compile an exhaustive list. As will be shown later, this arbitrary selection does not necessarily compromise the validity of the result.

Figure 4 shows a partial plot of the results on two of the file groups, the *executables* and the *C source code* groups. This is an example of how different file types consistently exhibit different compression ratios — the file types form distinct linear groups. It appears that division into groups is justified because of this.

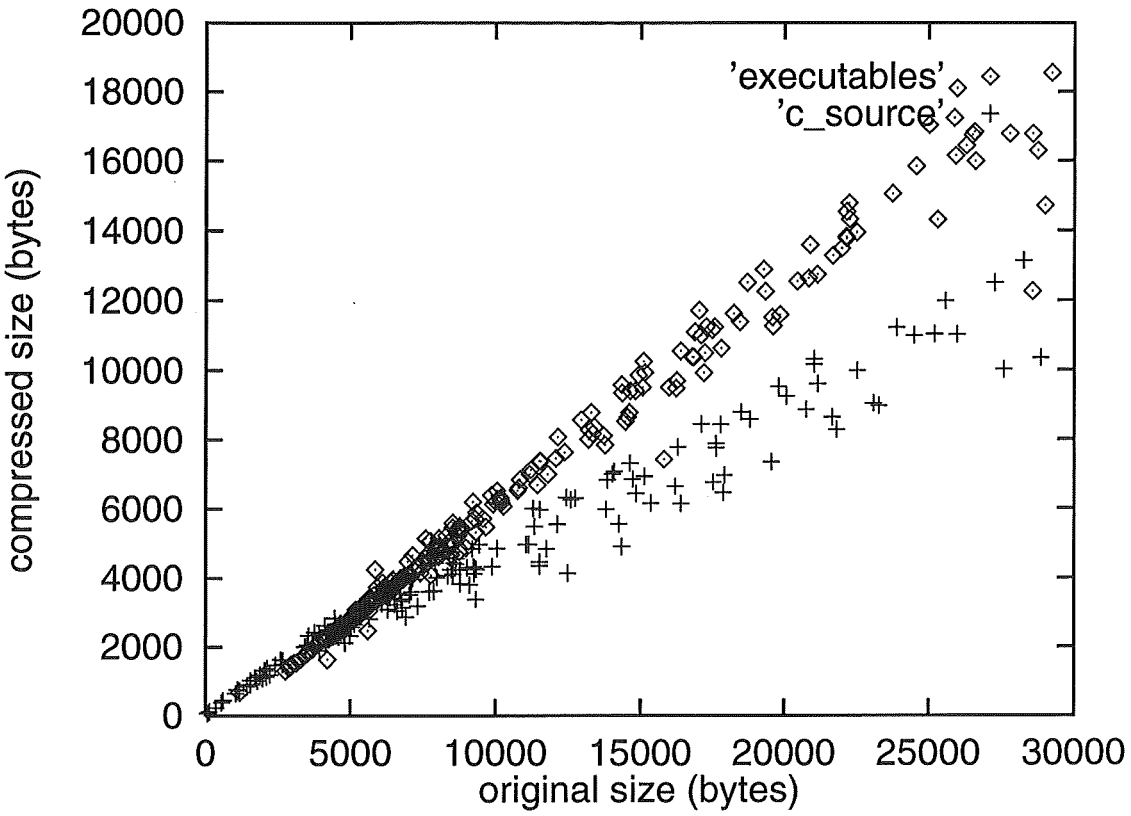

Figure 4: Results from applying the *compress* program to files of two different types. Each data point represents a file. The x coordinate is the size before compression ; the y coordinate is the size after compression.

For each group, we want to select a small number of representative files, with regard to compression. To do this, we compressed each file in the group with a number of compression methods. For each compression algorithm used, a simple scatter plot of file size before and after compression was obtained. Because files in the group had similar characteristics, the relationship was approximately linear. A straight line was fitted to the points using ordinary regression techniques. The slope of this line gives the average compression of that file set for that algorithm. An example of this is shown in Figure 5, which represents 39 English text files from the Project Gutenberg collection.

A file is then selected whose data point lies close to the regression line for all compression methods used. Such files can be said to be compressed typically by

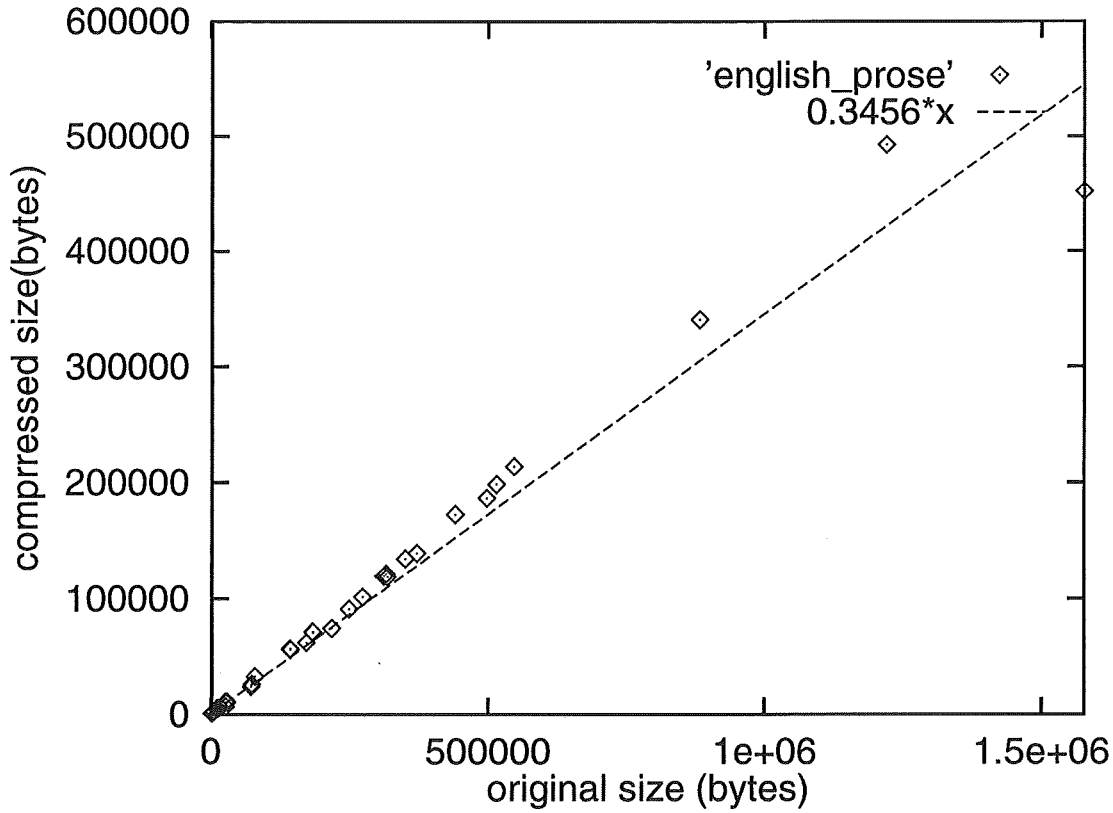

Figure 5: Regression plot showing the compression of several files from the Project Gutenberg collection. The compression program used to generate these results was an implementation of Wheeler's block coding algorithm (Wheeler, 1995).

a number of different algorithms. We claim that new algorithms are also likely to perform typically on files selected in this way. This claim is further discussed in section 5.

Data points are considered to be close to the regression line if the square of their normalised distance above or below the line is small. Squaring the distance from the line has the effect of emphasising the distance from the line. Summing these normalised distances across all algorithms used gives the combined distance — the files with the smallest sum are considered the best candidates for inclusion.

One possible problem with this method may occur when a very large size range is involved. For example, the largest files in a set may be 50 times larger than the smallest. In cases like this the larger files are likely to have larger residuals, that is, their data points are likely to fall further from the regression line (Chen & Stromberg, 1996). To compensate for this problem, the relative distance from the regression line was used to rank files, rather than the absolute distance. This approach is shown in Figure 6, where a is the absolute distance and a/b is the relative distance, for the given point.

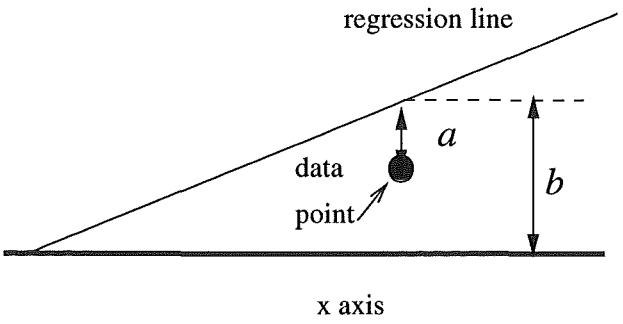

Figure 6: How rankings were obtained for each file. The absolute residual, a/b, was used.

5 Results

In this section the results of applying the methods detailed in the previous section to a number of file sets are given.

The Canterbury corpus is given in Figure 5. The files were divided into several groups as shown. The number of files in each group is given by N.

The *average correlation* column gives the correlation coefficient for the linear regression. Note that in the case of the CCITT test set, all files were the same size — the data points fall on a vertical line, so no correlation measurement is possible. The high numbers obtained in this column (almost all over 0.99) indicate that a linear model fits the data very closely. It also implies that adding more data (by increasing the size of the file sets) would add little more information. Thus file sets do not have to be extremely large to produce useful results.

The *best file* is the file with minimal normalised deviation from the regression line. This is effectively the file whose compression is close to the mean compression for that group. The selected file is the best representative of that group. Using this method has resulted in a small corpus relative to the total size of the files used to generate it. Effectively, all the files used are represented by the small selected subset.

In most cases, a file with very small sum of squares (that is, very close to average performance across all algorithms) was selected. The HTML group is less homogeneous than the other groups. HTML code can consist of a large number of control codes, making it similar to source code, or can consist mostly of ordinary text. This factor has led to the unusual results for the *html1* data set.

Results for a number of compression programs on the Canterbury Corpus are given in table 5.

5.1 Use of the Results

As mentioned in section 1, the results developed from these empirical techniques are likely to be useful to several groups of people. In this section we discuss the practical applications of our results. We also discuss the distribution of our results, which we consider critical for their full benefits to be realised.

File Set	Description	N	average correlation	best file	normalised deviation from line
guten_technical	Technical documents	15	0.992	world91b.txt	0.06016
guten_prose_a-g	English text	28	0.996	charble.txt	0.00204
guten_prose_h-z	English text	40	0.987	hydea10.txt	0.00566
guten_prose_all	English text	68	0.990	alice29.txt	0.00985
guten_poetry	English Poetry	5	0.998	plrabn12.txt	0.00085
html1	HTML	19	0.984	dcc.html	0.21987
manpages_1	UNIX Manual pages	502	0.995	alias.1	0.04536
shakesp	Plays	27	0.995	asyoulikeit	0.00014
sparc_exe	Executables	356	0.994	adb	0.00356
CCITT Test Set	Fax images	8	n/a	ptt5	0.00732
c_source	C source code	164	0.983	fields.c	0.00103
lisp	lisp source code	76	0.948	infix.lisp	0.00135

Table 4: The Canterbury Corpus

	pack	compress	DMC	gzip	bred	PPMC
act.html	5.63	4.68	3.39	3.00	3.06	2.85
alias.1	5.10	4.44	3.45	3.15	3.17	2.90
alice29.txt	4.61	3.27	2.75	2.86	2.54	2.22
asyoulikeit	4.85	3.51	2.96	3.12	2.83	2.50
fields.c	5.12	3.56	2.40	2.25	2.16	2.11
lcet10.txt	4.70	3.05	2.80	2.71	2.46	1.97
my-grammar.lisp	4.87	3.89	2.84	2.68	2.69	2.39
normal.xls	4.89	3.67	2.55	2.54	2.47	2.30
plrabn12.txt	4.58	3.37	3.03	3.24	2.88	2.37
ptt5.pm	1.08	0.24	0.20	0.28	0.21	0.20
rpcinfo	5.82	4.95	3.92	3.86	3.81	3.56
Average	4.66	3.52	2.76	2.70	2.58	2.31

Table 5: The Canterbury Corpus

6 Conclusion

A number of issues relevant to the evaluation of data compression algorithms have been discussed. These included the importance of noting and explicitly stating the versions of the files and programs used, methods of measurement, and the history and possibilities of misrepresentation in the area of data compression.

We have shown that current methods of evaluating data compression techniques, in particular the selection of appropriate test data, are questionable. We have proposed a new way of selecting test data, and used this method to develop a test corpus appropriate for the evaluation of lossless compression algorithms. This corpus will shortly become publicly available and we hope for a positive response from the data compression community.

Appendix A : Distribution

The home page for the Canterbury Corpus will be under **http://www.cosc.canterbury.ac.nz/ tim/corpu**

The files and programs discussed in this section are available through the Canterbury Corpus home page. In the near future we hope to distribute a CD containing these files — please direct enquiries to Tim Bell (`tim@cosc.canterbury.ac.nz`).

The distribution includes the following.

- Source code (ANSI C) and sample input files for the finite state automaton discussed in Section 3.4.
- All files in the Canterbury Corpus.
- The Calgary Corpus.

References

1996 (Sept.). *compression-faq.* Available by anonymous ftp from rtfm.mit.edu.

Abrahamson, David M. 1989. An Adaptive Dependency Source Model for Data Compression. *Communications of the ACM*, 77–83.

Bell, Timothy C., Cleary, John G., & Witten, Ian H. 1990. *Text Compression.* Prentice Hall.

Bryan, John. 1995. Compression Scorecard. *Byte*, May, 107–110.

Chen, Yeh-Ling, & Stromberg, Arnold J. 1996. Comments on Data Analysis in Software Experiments. *SIGICE Bulletin*, **22**(1).

Ferens, K., & Kinser, W. 1995. Adaptive Wavelet Subband Coding for Music Compression. *Page 460 of: Proceedings of the Annual IEEE Data Compression Conference.*

Frey, Brendan J., & Hinton, Geoffrey E. 1996. Free Energy Coding. *In: Proceedings of the Annual IEEE Data Compression Conference.*

G.Roelofs. 1996 (Oct.). *zlib home page.* from http://quest.jpl.nasa.gov/zlib/.

Inglis, Stuart, & Witten, Ian H. 1994. Compression-based Template Matching. *In: Proceedings of the Annual IEEE Data Compression Conference.*

Kiselyov, Oleg, & Fisher, Paul. 1994. Self-Similarity of the Multiresolutional Image/Video Decomposition: Smart Expansion as Compression of Still and Moving Pictures. *In: Proceedings of the IEEE Data Compression Conference.*

Park, Stephen K., & Miller, Keith W. 1988. Random number generators : good ones are hard to find. *Communications of the ACM.*

Shamoon, Talal, & Heegard, Chris. 1994. A Rapidly Adaptive Lossless Compression Algorithm for High Fidelity Audio Coding. *In: Proceedings of the Annual IEEE Data Compression Conference.*

Shannon, C. E. 1948. A Mathematical Theory of Communication. *Bell Systems Technical Journal.*

Slyz, Marko J., & Neuhoff, David L. 1996. Piecewise Linear Tree-Structured Models for Lossless Image Compression. *In: Proceedings of the IEEE Data Compression Conference.*

Teahan, W.J., & Cleary, John G. 1996. The Entropy of English Using PPM-Based Models. *In: Proceedings of the Annual IEEE Data Compression Conference.*

Wu, Xiaolin. 1996. An Algorithmic Study on Lossless Image Compression. *Pages 150–159 of: Proceedings of the Annual IEEE Data Compression Conference.*